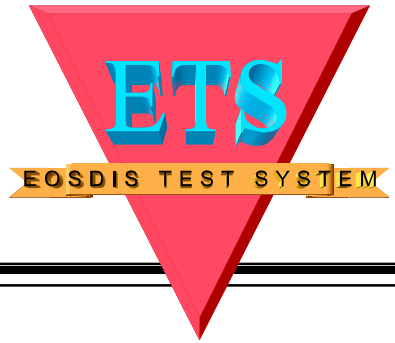
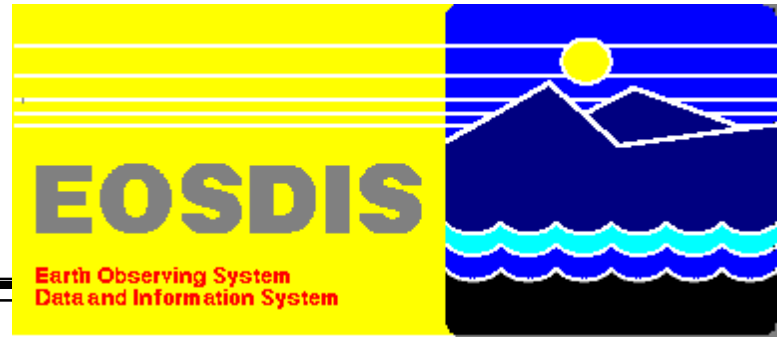




EOSDIS Test System (ETS) for PM-1 Support

Design Status Presentation

May 13, 1999

Review Purpose

Acquaint the user community with our current design status and provide for feedback into the SIMSS/PM-1 development effort.

Collaborative Development (1 of 2)
(Copied from the SIMSS SDR Presentation)

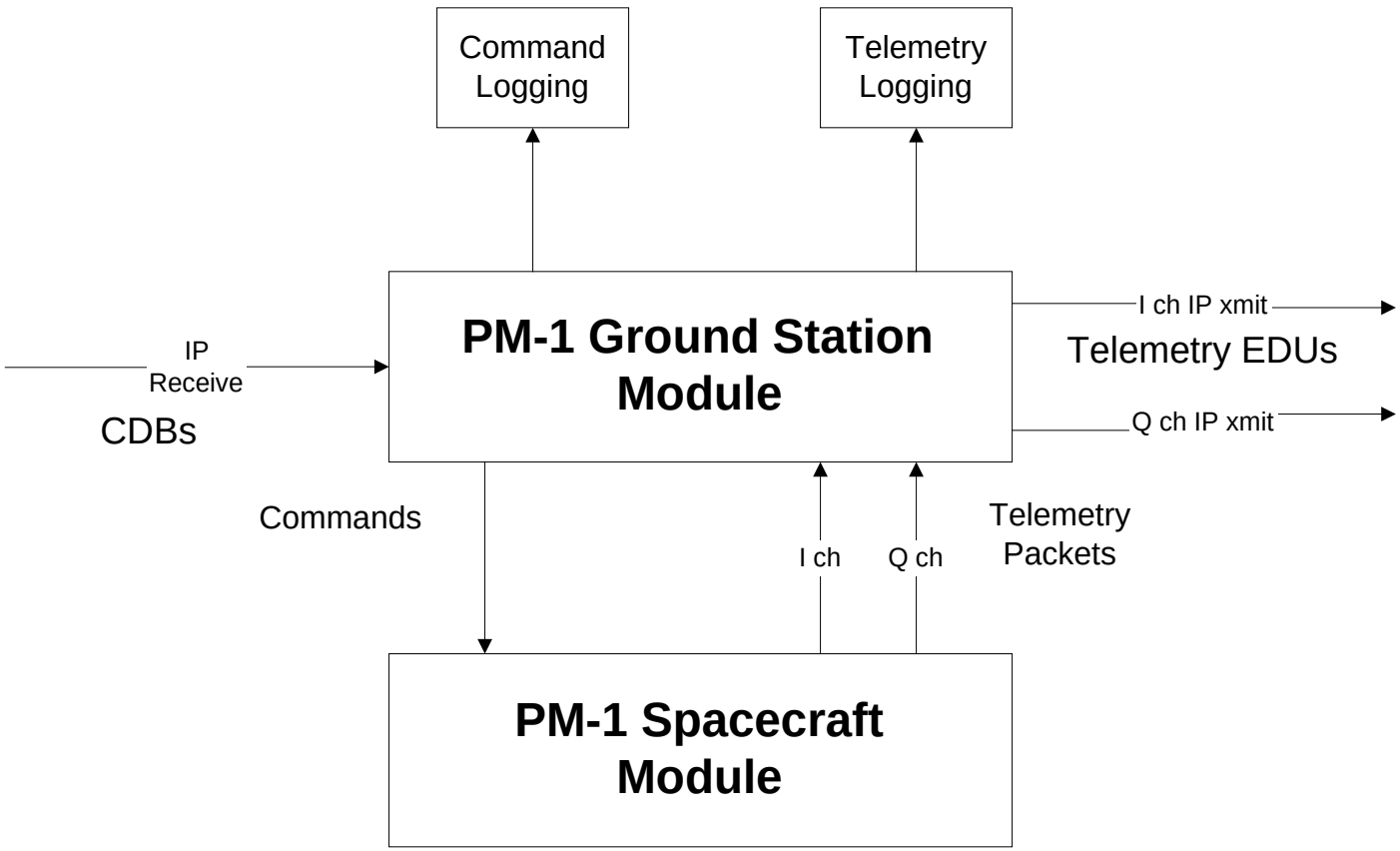
- ETS work funded by Mission Systems as CSOC SODA task (G936)
  - Work includes:
    - » development of PM-1 and future EOS simulators
    - » current support of Terra MPS
    - » upgrade of SCTGEN for PM-1 & beyond
- Separate SOMO-funded SODA task (G903) to enhance our suite of spacecraft simulators and test tools
  - Called Scalable Integrated Multi-mission Simulation Suite (SIMSS)
    - » spacecraft component (SC)
    - » network test tool (NeTT)

Collaborative Development (2 of 2)
(Copied from the SIMSS SDR Presentation)

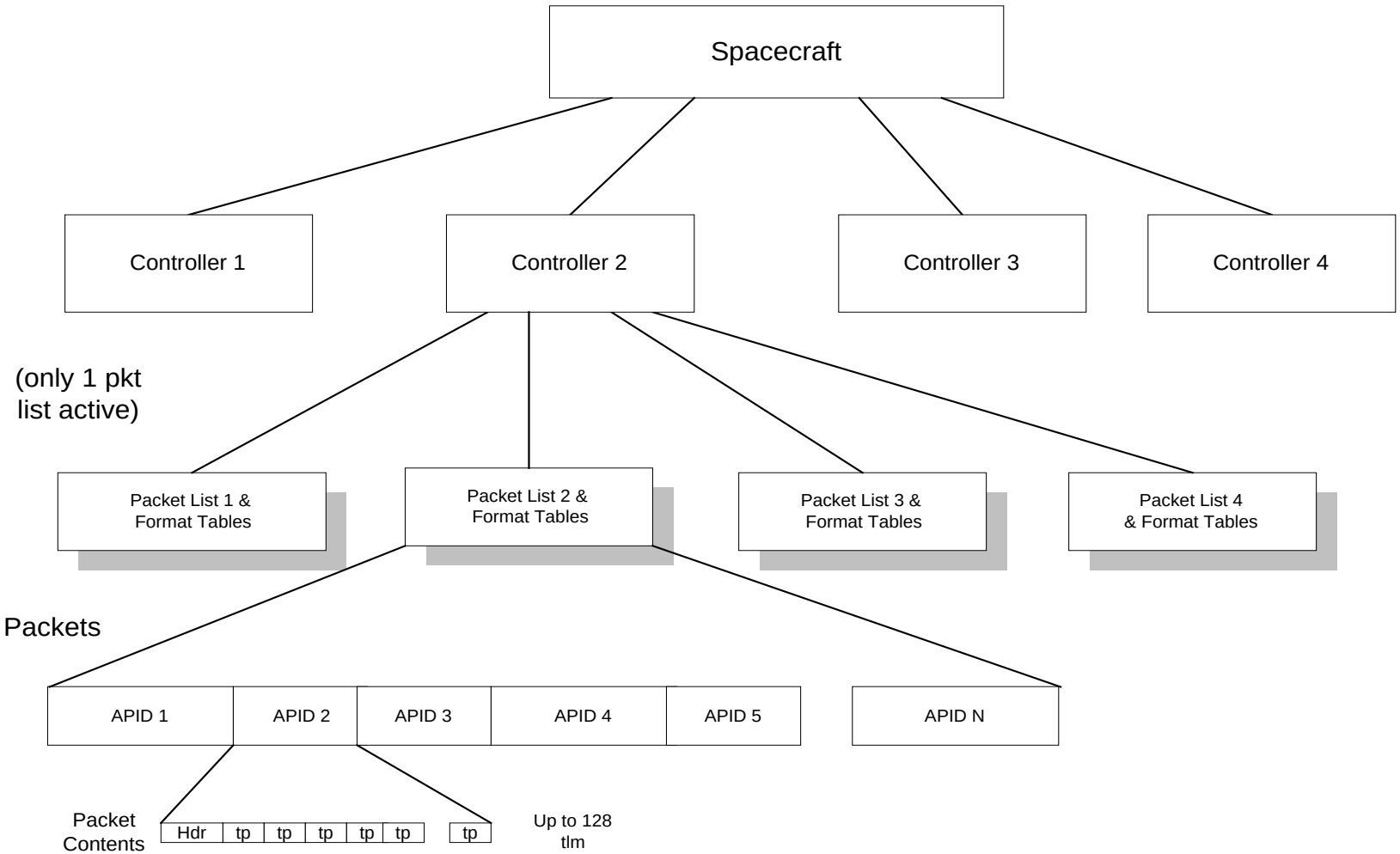
- The ETS PM-1 spacecraft simulator
  - Referred to as SIMSS / PM-1
  - Built on the SIMSS architecture and baseline objects
  - ETS developers adding PM-1 specific extensions to simulator and test tool

Simulator Design for PM-1

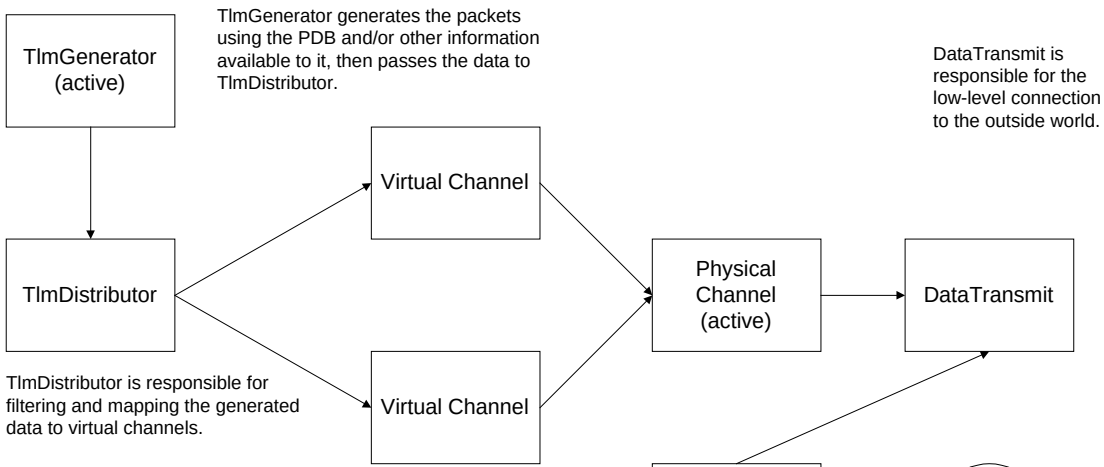
Overview


Command Logging
Telemetry Logging
PM-1 Ground Station Module
IP Receive
CDBs
I ch IP xmit
Telemetry EDUs
Q ch IP xmit
Commands
I ch
Q ch
Telemetry Packets
PM-1 Spacecraft Module

Telemetry Diagram



Telemetry Data Flow Diagram



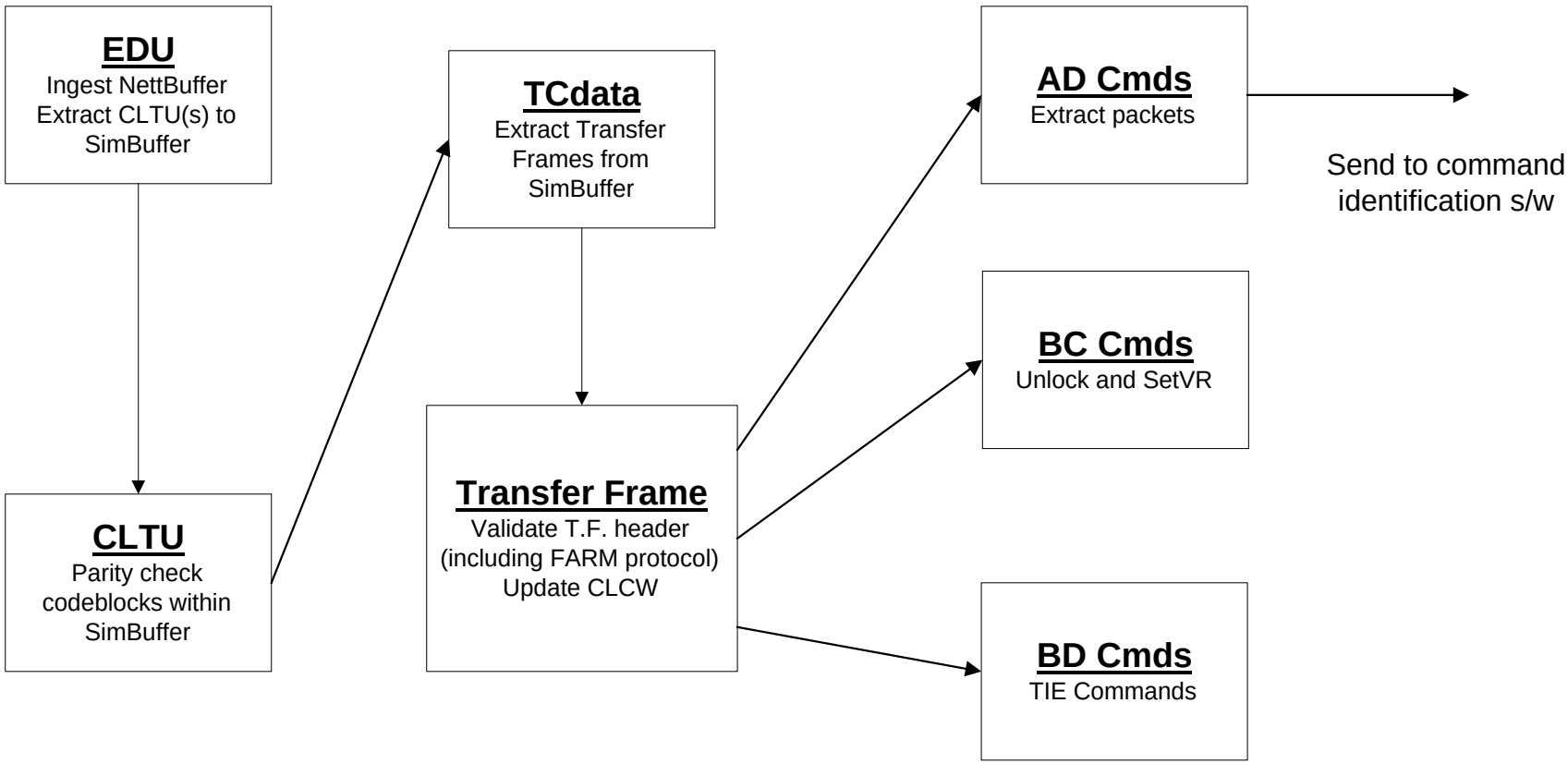
Telemetry Use-Case Description

Refer to

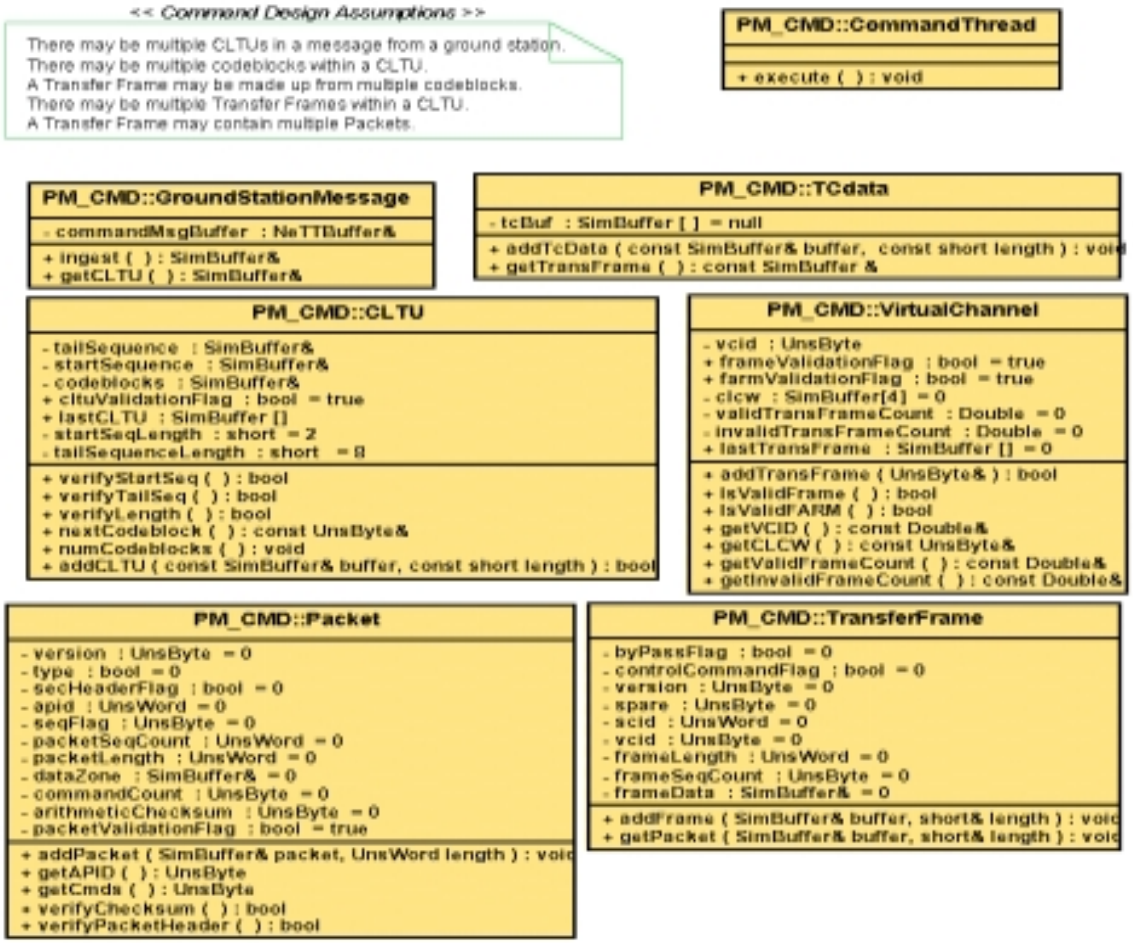
"Use-Case Textual Description for PM-1 Telemetry Generation"

Word document

Command Ingest Data Flow Diagram



Command Ingest Class Hierarchy



Command Ingest Reports

Refer to

“CMDIngestClassModel.doc”

Word document

Command Use-Case Description

Refer to

"Use-Case Textual Description for PM-1 Command Ingest"

Word document

Use-Case Descriptions

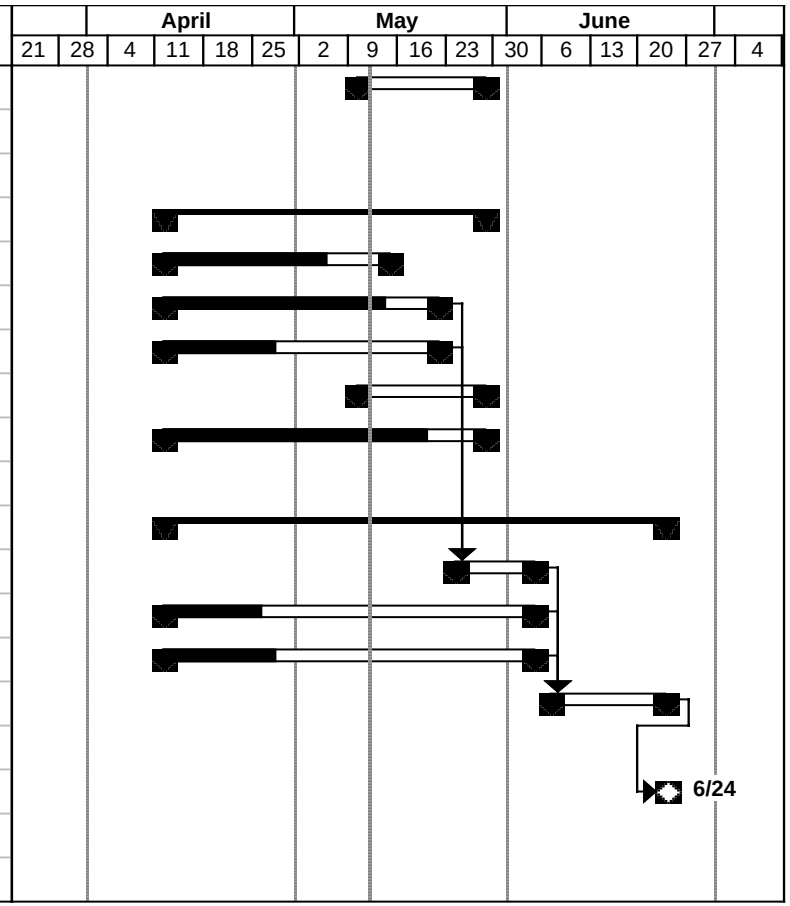
- Other Use-Case Descriptions to be produced later
  - command recognition
  - end-item command verification
  - memory load and dump
  - Format Table / Packet List load and switch

Functionality for Release 1.0

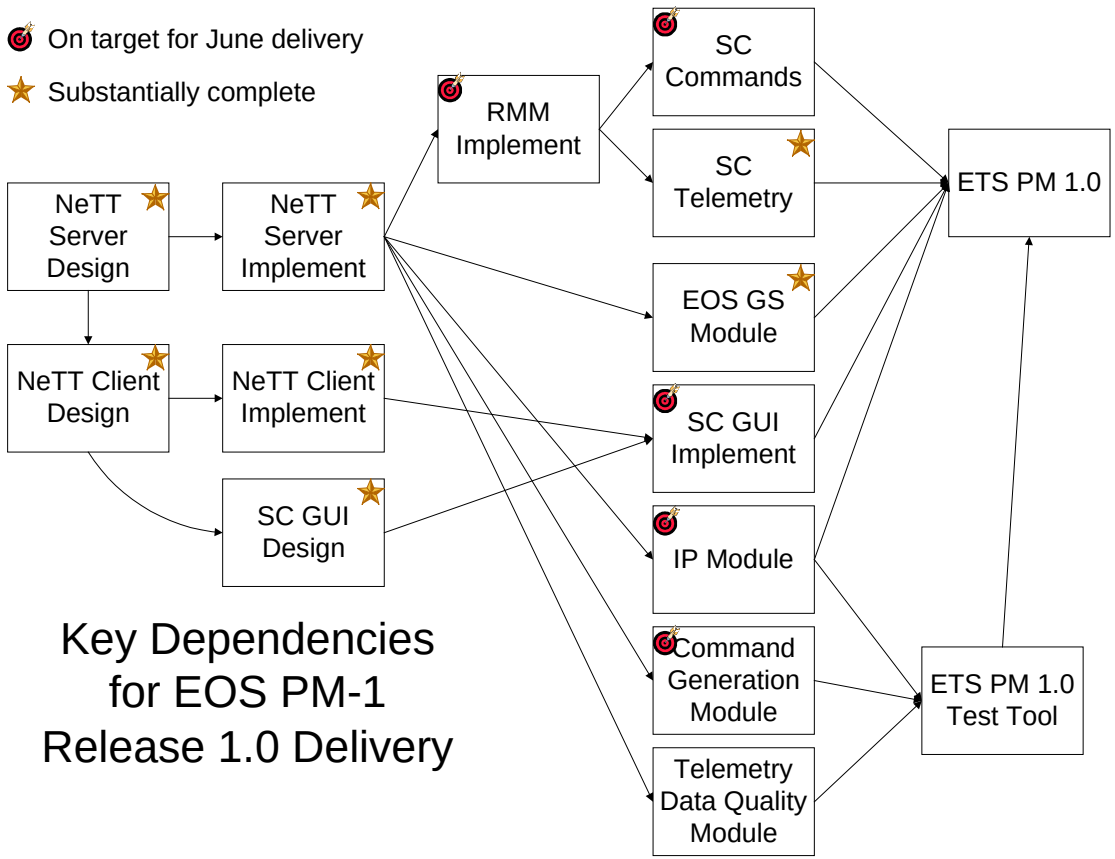
- Static telemetry in IP mode EDUs
- Command ingest
- Logging of Command Data Blocks and EDUs
- Maintain GMT and Spacecraft clock
- User Interface
  - Display telemetry status
  - Display event messages
  - Start and stop telemetry transmission

Development Schedule

ID	Task Name	Duration	Start
20	GUIs	15 days	Mon 5/10/99
21			
22			
23	**SC/PM-1 Subsystems**	**35 days**	**Mon 4/12/99**
24	Sim/RMM	25 days	Mon 4/12/99
25	Telemetry	30 days	Mon 4/12/99
26	Commanding	30 days	Mon 4/12/99
27	SC Main	15 days	Mon 5/10/99
28	GUI	35 days	Mon 4/12/99
29			
30	**Integration and Testing**	**53 days**	**Mon 4/12/99**
31	Integration/Integration testing	10 days	Mon 5/24/99
32	System Test Plan	40 days	Mon 4/12/99
33	User's Guide	40 days	Mon 4/12/99
34	System Test	13 days	Mon 6/7/99
35			
36	Delivery	0 days	Thu 6/24/99



Dependencies Matrix



Test Tools

- NeTT
  - Telemetry
    - » Delog and display by EDU
    - » Eventually use the PDB to display by parameter
  - Command
    - » Log in parallel with SIMSS/PM-1
    - » Delog and display by CDB
    - » Later on:
      - display by CLTU and Transfer Frame
      - Use the PDB to display by command
      - Use the PDB to generate valid PM-1 commands

Test Tools, continued

- Command Generator
- EDU Delogger
- Equipment at SOC (Bldg 25) for telemetry logging
- HexEdit

Needs List

- The PDB DFCD
- The PDB as flat files
- EDOS to EGS ICD updates
- preloaded Packet Lists and Format Tables
- Ada symbol to telemetry point physical address mapping
- Mnemonics of command counters that appear in telemetry
- Date for rigorous timing of telemetry packets
- Date for support of multicasting
- Formal points of contact within each organization
- Detailed, prioritized, requirements list for command subsystem

Information Sources

- ICD Between the EOS Common Spacecraft and the EOS Ground System (EGS) October 15, 1998 (TRW)
- Appendix Z
- TRW IOCs
- The MIT Website
  - Question / Answer Matrices
  - IOCs
  - ICDs

Information Sources, continued

- Flight Software Requirements Specification
- Command Allocation Document
- Telemetry Allocation Document
- EOS Command and Telemetry Handbook for the PM-1 Spacecraft, May 15, 1998 (TRW)

Use-Case Textual Description for PM-1 Telemetry Generation

Telemetry simulation for the PM-1 spacecraft is much different from AM-1. PM-1 telemetry is controlled by packet lists and format tables, to be explained below.

PM-1 telemetry is sent out by the SIMSS simulator as EDUs. Each EDU will consist of an EDOS Service Header (20 bytes) followed by one VCDU. The VCDU consists of a VCDU header, and a Data Unit Zone. The Data Unit Zone consists of one M_PDU Header and 208 bytes of telemetry information encapsulated in CCSDS packets. See Figure 6.2.1-1 on page 6-2 of the Space-to-Ground ICD for a picture of the VCDU Data Unit Zone.

The spacecraft contains four controllers (and the AIRS controller, to be added later). Each controller is responsible for a subset of the total telemetry generated by the spacecraft. For reference purposes they are the Command & Telemetry Controller (CTC), the G&NC Controller (GNCC), the Power Controller (PC), and the Instrument Support Controller (ISP).

The following discussion applies to each controller.

As noted above, the telemetry generated by each controller is controlled by packet lists and format tables. Each controller has four packet lists, of which only one is active at a given time. Which one is active is determined by operator entry. (TBD – in a later release the simulator may accept a ground command to set the active packet list.)

Telemetry packets generated by the instruments are not covered here. The amount and type (packet contents and lengths) have not yet been researched. It is TBD but currently assumed that a fifth controller will be added later to simulate instrument telemetry.

Packet lists appear to be allocated as follows. One for 16k telemetry (known as housekeeping), one for 4k telemetry (TDRS SA mode), one for 1k telemetry (TDRS MA mode), and one as backup to be loaded and used as necessary. It is planned by the FOT that a switch from one packet list to another will be performed via Stored Command just prior to contact with a ground station or TDRS. SIMSS can simulate this via operator entry. In the absense of any operator entry, SIMSS should default to 16k telemetry. Since SIMSS cannot tell from Packet List/Format Table contents which packet list is desired for the data rate selected, it is the responsibility of the operator to ensure that the correct packet list is selected for the data rate requested.

The contents of all packet lists and format tables will be supplied by EMOS.

A packet list contains from one to 16 entries. Each entry consists of three fields: (1) the number of the corresponding Format Table, (2) the Sample Period, and (3) the Slot Number.

- The Format Table defines all of the telemetry data that will go into the packet and thus the packet length.
- The Sample Period field defines the frequency of transmission of the packet using one of the 10 modulo counters. e.g. the Sample Period field contents will be the number of one of the modulo counters. The range of the field is 0 to 9.
- The Slot Number is the offset from the zero count of the modulo counter defined by the Sample Period. Its purpose is to even out the rate of telemetry generation. The maximum size of each packet list is 48 16-bit words.

Each controller has 16 Format Tables. Each Format Table contains a header and from one to 128 telemetry point definition entries. The header contains the memory dump flag (more later), a count of the number of telemetry points defined, and the packet APID. Each telemetry point definition entry contains the access type and the physical memory address of the data to be collected. The access type tells the controller the number of bytes (1 to 6) to collect for that telemetry point.

Aboard the spacecraft, the physical memory address is the address in that controller's memory where the first, or least significant, byte of data resides. SIMSS will fake this as follows. All objects that are simulating controllers will have to locate the telemetry node for every entry in all Format Tables. This involves finding an equality between the entry in the Format Table and the physical address stored in the telemetry node. An offline program will be used to add the physical addresses to all telemetry nodes when the PDB is translated for SIMSS use. It was originally thought that the same offline program could be used to perform the equality matching to determine telemetry mnemonics from physical addresses. However, if SIMSS achieves the level of fidelity needed to accept a table load of Format Tables or Packet Lists, as it is understood the customer desires, then the mapping will have to be done in realtime.

We have an external need for the "Ada symbol to physical address map" to be supplied to us by EMOS.

Another question involves telemetry points that are single bit in nature. It is being presumed that one and two bit flags will be collected together in a single byte or word for transmission. Aboard the spacecraft these will most likely occupy the same byte of memory. If they (some of the bits) are set/reset by a piece of hardware that is external to the controller, then there is no problem from the SIMSS point of view. Their address will be the base address of that byte. If, however, there are bits that are set independently by the controller, or bits that are given telemetry mnemonics, then the physical address of some mnemonics may not correspond to the address given in the Format Table but will be in the same byte/word that that address points to. The existance of these telemetry points is TBD.

It is known that a nearly identical problem was encountered and solved in the development of the Landsat simulator. Therefore the easiest solution may be to reuse as much of that design and code as possible.

Failing that, alternative solutions are possible. It may be necessary to have the offline PDB translator examine addresses for all parameters that are not an integral number of bytes in length and group those that reside at the same "base" address. Perhaps mnemonics will have to be invented to describe the group. The objective is to make it easy for the online software, which is time constrained, to assemble the bits into a byte or word to go into the packet, while still making it easy for the operator to set a telemetry point by mnemonic. Implementation details are TBD.

Next subject: Here is how the modulo counters work. There are ten modulo counters per controller aboard the spacecraft. They all count in lockstep, incrementing once every 125 milliseconds (msec) and roll back to zero when their respective modulo counts are reached. The following table gives the ranges of the counters. Notice that the mod_1 counter doesn't really count. It merely ticks every 125 msec.

counter	**range**
mod_1	0
mod_2	0-1
mod_4	0-3
mod_8	0-7
mod-16	0-15
mod_32	0-31
mod_64	0-63
mod_128	0-127
mod_256	0-255
mod_512	0-511

Example: If entry #3 of the packet list has 2 as its Sample Period and 3 as its Slot Number, this means that that packet will be generated every time the mod_4 counter reaches 3.

The following PDL is given as a guide to the coders of controller modules.

```
// INITIALIZATION
As soon as the current packet list is determined
For every entry in every Format Table referenced by the current packet list
        Locate the corresponding telemetry node in the PDB by comparing the physical address to
        addresses in the PDB until a match is found.
        Output a warning message if no match is found.
EndFor
For each entry in the current Packet List
        Initialize the packet sequence counter
        Build the static portions of the packet leaving placeholders for any dynamic portions
        This includes:  Version number, Secondary Header Flag, Type, APID, Seq. Flags, and Packet Data
        Length
EndFor

// NORMAL OPERATIONS
Do Forever
        Wait until the time interrupt indicating it is time to build the next packet(s)
        For each packet to be built
                Increment the packet sequence counter
                //  The sequence counter rolls back to zero when it reaches 16384
                Insert the current time into the Secondary Header
                Index to the proper Format Table
                For each entry in that Format Table
                        If the corresponding telemetry point has changed
                                Insert the telemetry value into the packet
                        EndIf
                EndFor
                Send the packet to the Ground Station module
        EndFor
EndDo
```

The controller module simulating the CTC will construct and send a TIE packet each time the mod_8 counter recycles to zero.

Aboard the spacecraft, packets are assembled into VCDUs, a CADU sync pattern is prepended, Reed-Solomon check symbols are attached, randomization is performed, and the result is transmitted. The EDOS Ground Station element removes the randomization, strips the CADU sync and Reed-Solomon check symbols from data block, and transmits the VCDU to EMOS. The SIMSS simulator will build EDUs, bypassing the VCDU building steps. Randomization will not be performed.

The "Ground Station" module will take packets from the controller modules and assemble them into EDUs. When assembled, EDUs will be sent over the I, Q, or I and Q channels simultaneously, depending upon what the operator has selected. If I and Q transmission is selected, both channels will contain the exact same data, transmitted at the same time. Note that this is EPGS 16k mode of operations.

EDUs are built as telemetry packets are received. If the packet received will not all fit into the current EDU, as much data as will fit is put into the EDU and it is transmitted. The remaining packet data will go into the next EDU to be built. In this case, the M_PDU First Header Pointer of that next EDU will be set to one byte beyond the end of the partial packet. i.e. it points to the first packet header in the EDU. Note also that the maximum packet size is 256 bytes. It can take three EDUs to transmit a single packet.

CLCW EDUs are transmitted at the same rate as data EDUs; one CLCW EDU for each data EDU. If I and Q channels are both active, the exact same CLCW EDU is transmitted over both channels simultaneously. There are two CLCWs (see the command ingest description). They are transmitted in round-robin fashion. eg data EDU and spacecraft CLCW EDU, next data EDU and instrument CLCW EDU, next data EDU and spacecraft CLCW EDU, next data EDU and instrument CLCW EDU, etc.

We have also been asked to simulate the operation of the "ping-pong" buffers aboard the spacecraft. These are fixed sized buffers with the following hardware attributes/constraints:

- reside in the CTC and provide an intermediary holding point for packets that are destined for transmission to the ground
- fixed size of 256 bytes
- Cannot split arriving packets across buffers
- Cannot leave arriving packets on the 1553 bus
- If data arrives to quickly to be removed from the buffers to VCDUs (aboard the spacecraft) the buffers overflow and data is lost.

This means that the following scenario is possible: If buffer A is partially full and a packet arrives that will not fit, it goes into buffer B. If the next packet arriving will fit, it goes into buffer A. When EDUs are being filled, buffer A is emptied before any data is removed from buffer B. This means that data will arrive at the ground out of time order. That is what the EMOS people want us to simulate so they can test their software for re-arranging packets by APID.

There are a couple of questions to which we need to get answers: (1) are the buffers circular? If so, at a high data rate it might be a long time before any data is removed from buffer B. (2) Once all data is removed from buffer A and the switch is made to buffer B, does new incoming data go into buffer B, or A?

The following PDL is supplied to assist the coders of the "Ground Station" module. This PDL does not cover the following cases:

- ping-pong buffer simulation
- Q-channel 256k playback
- Fill CADUs – It is not known whether EDOS will remove fill CADUs or send them on to EMOS.

```
// Initilization – Some is TBD
Construct static portions of EDUs
Do Forever
        As packets arrive from the Controllers place them into the current EDU being built
        As packets arrive place them into the current EDU until 208 bytes have been filled
        If the current packet fills the current EDU
                Transmit the EDU on the I and/or Q channels, as per the current configuration
                Start a new EDU setting the First Header Flag to point to the first full packet in the EDU
        EndIf
EndDo
```

One unknown in this is whether the Primary VCDU Header is included in the EDU or just the M_PDU subheader.

QUESTIONS:

Is the assumption of the allocation of packet lists near the top of this memo correct? e.g. one for 16k telemetry, one for 4k telemetry, etc.

Are there telemetry points that are not an exact multiple of 8 bits in size?

How much of the CADU is preserved in the EDU? It is assumed that only the CADU sync pattern and the Reed-Solomon check symbols are removed. Is the VCDU header transmitted to EMOS or is it removed by EDOS?

We are presuming one VCDU per EDU. Is this correct?

Are our assumptions about the working of the ping-pong buffers correct? Are they circular? How is the switch between buffers made?

It is assumed that SIMSS will have to complete EDUs with fill data occasionally in order to keep the bandwidth filled. Is this correct?

Under certain conditions the spacecraft transmits CADUs with nothing but fill data, in order to maintain the bandwidth. Is this correct? If so, does EDOS discard fill CADUs or send them to EMOS?

```
======================================================================
                  Report: Class Model Report
                  System: PMCMD
                  By    : equintin
                  Date  : Mon - May 10, 1999
======================================================================

#########################
    CLD_Class Symbols
#########################


--------------------------------
Class :     CLTU
Package :   PM_CMD
--------------------------------

Description: The CLTU class provides processing of the Command Link Transmission
Unit.

ATTRIBUTES:
- - - - - -
Name                        Type                        Visibility
----                        ----                        ----------
tailSequence                SimBuffer&                  Internal
startSequence               SimBuffer&                  Internal
codeblocks                  SimBuffer&                  Internal
cltuValidationFlag          bool                        External
lastCLTU                    SimBuffer []                External
startSeqLength              short                       Internal
tailSequenceLength          short                       Internal


OPERATIONS:
- - - - - -
Name                  Return Type           Parameters
----                  -----------           ----------
verifyStartSeq        bool                     (  )
Description: Verify that the start sequence is correct.

verifyTailSeq         bool                     (  )
Description: Verify that the tail sequence is correct.

verifyLength          Unsbyte &                   (  )
Description: Verify that total CLTU length minus the start and tail sequences is
a multiple of codeblock length.

nextCodeblock         const UnsByte&                 (  )
Description: Returns pointer to the start of the next codeblock in the CLTU.

numCodeblocks         void                     (  )
Description: Returns the number of codeblocks in the CLTU.

addCLTU               bool                     ( const SimBuffer& buffer, const
short length )
Description: Copies the next CLTU into the start sequence, codeblock, and tail
sequence buffer areas.
```

Class : Codeblock
Package : PM_CMD

Description: This class contains methods for processing a codeblock.

ATTRIBUTES:

Name	Type	Visibility
codeblock_p	UnsByte &	Internal
codeblockValidationFlag	bool	External

OPERATIONS:

Name	Return Type	Parameters
addCodeblock	bool	( )

Description: Define current codeblock.

verifyParity	bool	( )

Description: Verify that the BCH parity byte matches the calculated parity. Also verify that the fill bit in the parity byte is zero.

getTCdata	bool	(UnsByte &)

Description: Copy the codeblock data into the specified buffer location.

Class : CommandThread
Package : PM_CMD

Description: This class defines the Command processing thread for the Spacecraft (SC) Module.

OPERATIONS:

Name	Return Type	Parameters
execute	void	( )

Description: This is the execution loop for the Command thread.

Class : GroundStationMessage
Package : PM_CMD

Description: This class buffers command messages from a Ground Station.

ATTRIBUTES:

Name	Type	Visibility
commandMsgBuffer	NeTTBuffer&	Internal

OPERATIONS:

Name	Return Type	Parameters
ingest	SimBuffer&	( )

Description: Read in the next command message from the Ground Station interface.

Name	Return Type	Parameters
getCLTU	SimBuffer&	( )

Description: Return a pointer to the next CLTU in the current command message buffer. If there are no more CLTUs in the buffer, returns a null pointer.

Class : Packet
Package : PM_CMD

Description: This class defines the CCSDS Packets used by PM-1.

ATTRIBUTES:

Name	Type	Visibility
version	UnsByte	Internal
type	bool	Internal
secHeaderFlag	bool	Internal
apid	UnsWord	Internal
seqFlag	UnsByte	Internal
packetSeqCount	UnsWord	Internal
packetLength	UnsWord	Internal
dataZone	SimBuffer&	Internal
commandCount	UnsByte	Internal
arithmeticChecksum	UnsByte	Internal
packetValidationFlag	bool	Internal

OPERATIONS:
- - - - - -

Name	Return Type	Parameters
addPacket	UnsByte	(SimBuffer& packet, UnsWord length)

Description: Initializes the packet header field attributes and data zone pointer.

getAPID	UnsByte	( )

Description: Returns the APID of the packet

getCmds	bool	( )

Description: Return the command count from secondary if applicable or zero otherwise.

verifyChecksum	bool	( )

Description: Verify the Arithmetic Checksum if present. Returns true if checksum is correct or not applicable.

verifyPacketHeader	bool	( )

Description: Verify that packet header fields are valid.

Class : TCdata
Package : PM_CMD

Description: This class is used to collect the codeblock telecommand data stripped of parity bytes.

ATTRIBUTES:
- - - - - -

Name	Type	Visibility
tcBuf	SimBuffer [ ]	Internal

OPERATIONS:
- - - - - -

Name	Return Type	Parameters
addTcData	void	(const SimBuffer& buffer, const short length)

Description: Copy the specified bytes from the codeblock into a telecommand data collection buffer.

getTransFrame	const SimBuffer &	( )

Description: Returns pointer to the next transfer frame in the telecommand data buffer.

Class : TransferFrame
Package : PM_CMD

Description: This class defines a generic Transfer Frame.

ATTRIBUTES:
- - - - - -

Name	Type	Visibility
byPassFlag	bool	Internal
controlCommandFlag	bool	Internal
version	UnsByte	Internal
spare	UnsByte	Internal
scid	UnsWord	Internal
vcid	UnsByte	Internal
frameLength	UnsWord	Internal
frameSeqCount	UnsByte	Internal
frameData	SimBuffer&	Internal

OPERATIONS:
- - - - - -

Name	Return Type	Parameters
addFrame	void	(SimBuffer& buffer, short& length)

Description: Initialize transfer frame header and data fields to current transfer frame.

getPacket	void	(SimBuffer& buffer, short& length)

Description: Returns a pointer to a buffer and a buffer length corresponding to the next packet in the transfer frame. Returns null pointer and zero length if a packet cannot be returned.

Class : VirtualChannel
Package : PM_CMD

Description: This class provides FARM-1 processing for transfer frames in a given virtual channel.

ATTRIBUTES:

Name	Type	Visibility
vcid	UnsByte	Internal
frameValidationFlag	bool	External
farmValidationFlag	bool	External
clcw	SimBuffer[4]	Internal
validTransFrameCount	Double	Internal
invalidTransFrameCount	Double	Internal
lastTransFrame	SimBuffer []	External

OPERATIONS:

Name	Return Type	Parameters
addTransFrame	bool	(UnsByte&)

Description: Defines the current transfer frame for the virtual channel.

IsValidFrame	bool	( )

Description: Validates the transfer frame header fields.

IsValidFARM	UnsByte	( )

Description: Validates that the frame sequence number complies with the Frame Acceptance and Reporting Mechanism (FARM) -1 protocol.

getVCID	const Double&	( )

Description: Returns the virtual channel identifier for this object.

getCLCW	const UnsByte&	( )

Description: Returns the contents of the Command Link Control Word (CLCW)

getValidFrameCount	const Double&	( )

Description: Returns the valid transfer frame count

getInvalidFrameCount	const Double&	( )

Description: Returns the invalid transfer frame count

==========
This Report Generated By GDPro
==========

Use-Case Textual Description for PM-1 Command Ingest

In IP mode, commands come from EMOS as Command Data Blocks (CDBs.) A CDB consists of a 24-byte Ground Message Header (GMH) followed by up to 6000 bytes of command information. The command information consists of RF Acquisition Sequence data followed by two or more CLTUs. All data is in NRZ-L

The acquisition sequence consists of a minimum of 128 bits of alternating ones and zeros, beginning with a one.

The length field of the GMH gives the total length of the message in bytes, including the GMH length. Therefore 24 must be subtracted from that length to get the actual data length. The fields of the GMH that need to be validated may be determined from the EDOS-EGS ICD.

After validating and discarding the GMH, the command ingest software must parse the message for CLTUs.

A CLTU consists of:

- a 16-bit Start Sequence (EB90)
- one to 104 codeblocks (8 bytes each)
- an 8-byte Tail Sequence (C5C5 C5C5 C5C5 C579)

A critical NO-OP is required by the TIE to ensure synchronization. Therefore, the first CLTU will contain a critical NO-OP command only. It is immediately followed by another CLTU (with any type of command) with no intervening RF acquisition sequence. It appears that every CLTU is preceded by a critical NO-OP command CLTU. This may depend upon whether commanding is continuous. If there are breaks between commands, the byte pattern will be: acquisition sequence, critical NO-OP CLTU, command CLTU, repeat.

The critical NO-OP CLTU consists of the Start Sequence, one codeblock containing the Transfer Frame header and NO-OP command, and the Tail Sequence

Notice that CLTUs can be much bigger than AM-1 CLTUs were.

If requested, send the CLTU for display. TBD: Save the latest CLTU for later display? It has been stated that the last command received should be available for display to the operator at any time prior to shutdown of the simulator.

If requested, log the CLTU to disk file.

Extract codeblocks from the CLTU and assemble them into Transfer Frames (TF). Perform a BCH parity check on each codeblock as it is extracted. A BCH error in any codeblock results in the entire CLTU being discarded and an error message sent to the operator. This parity check may be disabled by the operator.

It appears there may be multiple Transfer Frames per CLTU. Since the Transfer Frame header contains the VCID, it is possible that Transfer Frames destined for spacecraft and instrument may be mixed.

After the Transfer Frame is assembled compare the actual TF length to the value given in the TF header. Set the CLCW status if the length is incorrect. This check may be disabled by the operator.

Note that there are two CLCWs. One is for spacecraft commands; the other is for instrument commands. They are differentiated by VCID. The information in the CLCWs is maintained for the telemetry transmit thread, which, in the spacecraft, appends a CLCW to every VCDU transmitted. The CLCWs are transmitted in round-robin fashion – spacecraft, instrument, spacecraft, instrument, etc. (The simulator telemetry thread will transmit a CLCW EDU with every telemetry EDU sent. See the telemetry description for more detail.)

Check the following fields of the Transfer Frame header: version, S/C ID*, bypass and control flags*, and virtual channel ID*. Errors in the three that are asterisked result in status being placed in the CLCW.

Note that there is no Transfer Frame Error Control (TFEC) field. However, many Transfer Frames have an arithmetic checksum in the secondary header. Details are available in the Space to Ground ICD and in TRW IOC memos. Error processing is TBD.

There are three types of Transfer Frames. Type AD Transfer Frames contain spacecraft or instrument commands and are subject to the FARM frame sequence count sliding window check. Note that there are two frame sequence counters, one for each VCID. Spacecraft Transfer Frames have VCID 0 (zero), instrument Transfer Frames have VCID 1 (one). Receipt of a Type AD Transfer Frame whose frame sequence count does not equal the Next Expected Frame Sequence Number (the Report field value of the corresponding CLCW) result in that Virtual Channel going into lockout mode. When a Virtual Channel is in lockout, no Type AD Transfer Frames are accepted until a Type BC Unlock command is received.

Type BD Transfer Frames contain TIE Critical commands. (This includes the critical NO-OP described above.) There may be only one TIE Critical command per BD Transfer Frame.

Type BC Transfer Frames contain Control Commands. There are only two Control Commands, UNLOCK (unlock the FARM lockout) and SET V(R) (set the Next Expected Frame Sequence Number). Again, these may have VCID equal zero for spacecraft, or equal one for instrument. There may be only one Control Command per BC Transfer Frame.

After command ingest, packets are extracted from type AD Transfer Frames and are passed to the command recognition software.

It appears that there may be multiple packets per type AD Transfer Frame. Since the Transfer Frame header contains the VCID, all packets in a Transfer Frame will be destined for either spacecraft or instrument.

Type BC Transfer Frames are perhaps best handled within the command ingest software as their effect is local.

Further processing of type BD Transfer Frames is to be determined.

The following are items for the command recognition software to be developed later. (These are placed here for informational purposes):

1) the ultimate destination of a command packet (spacecraft or which instrument) is determined by the APID, which is contained within the packet header.

2) It is known that some commands (AIRS has been identified) are longer than 48 bits. These commands arrive in two parts and must be reassembled before they may be acted upon. The implications for the simulator depend upon the amount of fidelity aspired to. They are to be determined.

3) All spacecraft and instrument commands are 48 bits (or fewer) long so that they may fit into a Stored Command Sequence slot. There are multiple spacecraft commands per packet. It is believed that there may be multiple instrument commands per packet.

Most of the preceding discussion may be found in a slightly different form in Section 5.1.2.4, Command Processing, of the PM-1 Flight Software Requirements Specification. The PM-1 space-to-ground ICD (TRW document D22262) contains a wealth of detail in a somewhat confusing format.

NOTICE: There are TRW IOCs that modify the Space-to-Ground ICD.

QUESTIONS:

Is every CLTU preceded by a critical NO-OP command CLTU?

It appears there may be multiple Transfer Frames per CLTU. Is this true? If so, will Transfer Frames destined for spacecraft and instruments be mixed in a single CLTU?

What error processing should be performed on detection of a checksum error? Alert the operator, of course. Discard the Transfer Frame?

Is there the possibility of multiple packets in a type AD Transfer Frame?